How thread-safe is the Windows Runtime PropertySet object?

devblogs.microsoft.com/oldnewthing/20201120-00

November 20, 2020

Raymond Chen

A customer was looking for a thread-safe collection type, and they found the Windows Runtime `PropertySet` object. Is this thing thread-safe?

Yes, the `PropertySet` object is thread-safe.

However, it still may not be what you want.

The `PropertySet` object is thread-safe in the sense that all of its operations are atomic. Concurrent usage from multiple threads will be consistent with some sequential order.

For example, if two threads both attempt to insert different values into the property set under the same key, it will be "last writer wins" with some choice of "last". If one thread attempts to insert a value under a key at the same time another thread attempts to retrieve the value under that key, the reader will either get the old value or the new value, not some weird in-between value.

That said, the available operations on a `PropertySet` may not be sufficient for your desired concurrent usage.

Here's what you can do with a `PropertySet`, or more generally, any `IMap<K, V>`:[1]

- Obtain the number of items by requesting the `Size` property.
- Empty the collection by calling `Clear()`.
- Check if a key is present by calling `HasKey()`.
- Look up the value associated with a key by calling `Lookup()`.
- Add or update an item by calling `Insert`. The return value tells you whether an existing item was replaced, or a new item was created.
- Remove an item by calling `Remove`.

That's it. In particular, you don't have methods like `GetOrCreate` which atomically retrieves an existing value or manufactures one if it doesn't exist yet. Or `InsertIfNew` which atomically creates a new value but doesn't modify any existing one. If you need

operations like those, then you'll have to create them yourselves, and that means adding your own lock around the `PropertySet`.

At which point, maybe you realize the thread-safety of a `PropertySet` doesn't buy you much, seeing as you're going to need a lock anyway. You may as well use a `std::map` or some other more convenient associative container.

[1] These are the methods available at the ABI level. Language projections may synthesize other methods out of these methods, but if those other methods involve multiple calls, then atomicity is lost.

Raymond Chen

Follow